Cupi2

Buscando nuevas maneras de enseñar a programar...

Manual de Usuario
PLUGIN ROBOT MOVIL

Tabla de Contenidos

1. REQUERIMIENTOS 3
   1.1 Hardware 3
   1.2 Software 3
2. INSTALACIÓN 3
3. FAMILIARIZACIÓN CON LAS VISTAS DEL PLUGIN 4
   3.1 Explorador del entrenador 5
   3.2 Vista de la gráfica del mundo del robot 5
      3.2.1 Principales características y restricciones del mundo del robot 6
      3.2.2 Botones de la vista 6
   3.3 Vista de los contenidos del nivel 11
   3.4 Vista de las precondiciones del nivel 12
   3.5 Vista de los enunciados de los retos 12
   3.6 Vista de los tips de los retos 13
   3.7 Vista de los mensajes del robot 13
   3.8 Botón para la ejecución de las soluciones de los retos predefinidos 13
   3.9 Vistas adicionales 14
4. TRABAJANDO CON LOS RETOS DE ENTRENAMIENTO PREDEFINIDOS 14
   4.1 Creando una nueva solución 14
   4.2 Ejecutando la solución de un reto 15
   4.3 Haciendo debug a la solución de un reto 16
   4.4 resolviendo problemas con la ejecución de soluciones 18
5. CREANDO UN NUEVO PROYECTO 18
6. CAMBIANDO LAS PREFERENCIAS DEL PROYECTO 19
7. API DEL ROBOT 21
   uniandes.cupi2.robotMovil.kernelCliente Class Robot 21
      Robot 24
      Robot 24
      cargandoCubo 25
      colorDelCuboCargado 25
      contactoConCubo 25
      obtenerDireccion 25
      obtenerX 26
      obtenerY 26
      tieneRobotAlFrente 26
      tieneRobotALaIzquierda 26

tieneRobotALaDerecha 26
tieneCuboAlFrente 27
tieneCuboALaIzquierda 27
tieneCuboALaDerecha 27
tieneMuroAlFrente 27
tieneMuroALaIzquierda 28
tieneMuroALaDerecha 28
limiteAlFrente 28
limiteALaIzquierda 28
limiteALaDerecha 28
marcaDelCuboCargado 29
agregarMensaje 29
definirVelocidad 29
soltarCubo 29
mover 29
rotarALaIzquierda 30
rotarALaDerecha 30
tomarCubo 30
prender 30
estaPrendido 30
nombreDelRobot 30
colorDelRobot 31
colorDelCuboCargadoCadena 31
8. CRÉDITOS 31
9. REFERENCIAS 31

1. REQUERIMIENTOS

1.1 Hardware

Los mismos de la plataforma Eclipse.

1.2 Software

- ✓ Sistema operativo Windows
- ✓ Eclipse: 3.0.0 o superior
- ✓ JDK 1.5 o superior

2. INSTALACIÓN

1) Baje el plugin del sitio: http://cupi2.uniandes.edu.co/cursos/apo1/nivel3.htm
2) Cierre la plataforma Eclipse (en caso de que la tenga abierta).
3) Descomprima el archivo "co.edu.uniandes.cupi2.E2_robot_movil_1.0.zip" que bajo en el punto anterior en la carpeta de plugins de Eclipse. El path de esta debe ser ../Eclipse/plugin.
4) Abra la plataforma Eclipse. Eliga En el menú `Windows>Open Perspectiva>Other` la perspectiva "Robot Movil". Si le aparece una ventana similar a la siguiente, la perspectiva ha sido instalada con éxito.

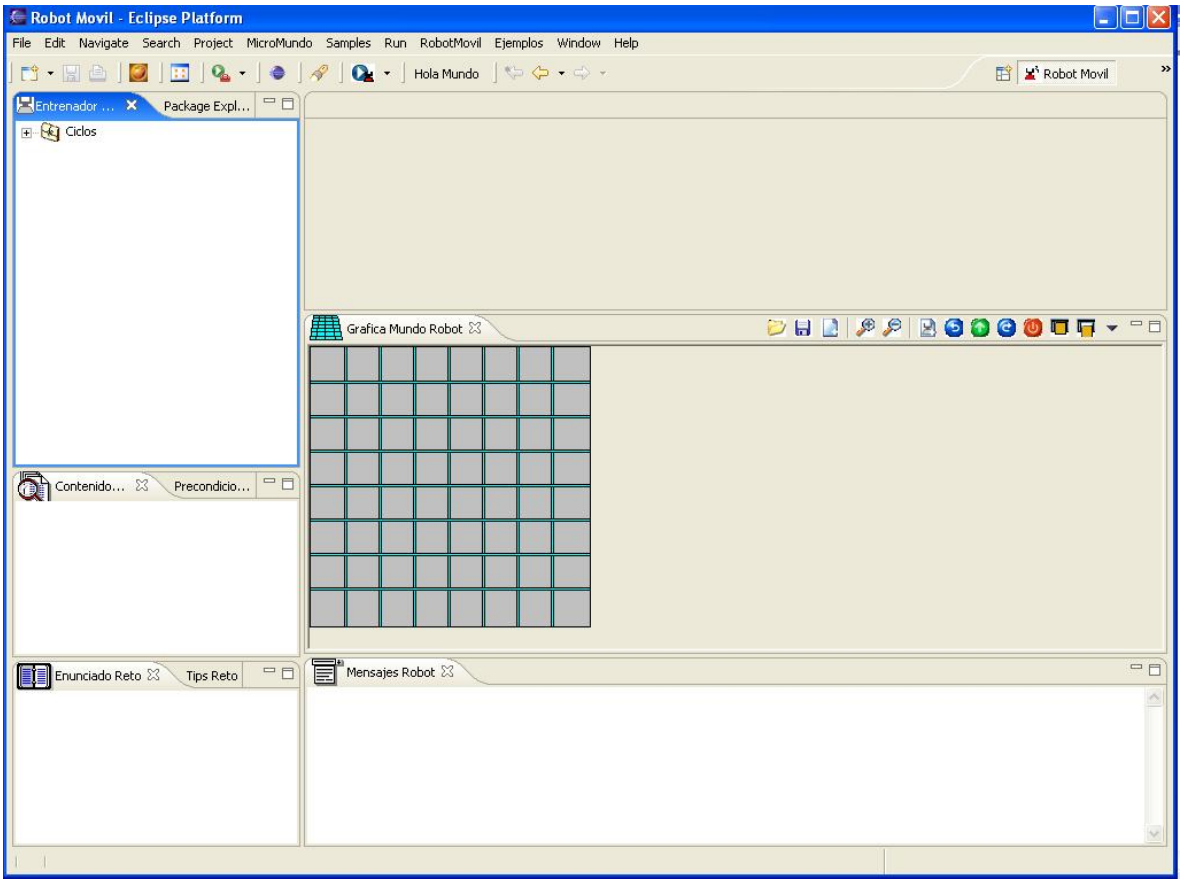


Figura 1. Pantalla inicial de la perspectiva

3. FAMILIARIZACIÓN CON LAS VISTAS DEL PLUGIN

El plugin consta de 7 vistas y un botón en la barra de herramientas, los cuales pueden ser apreciados en la vista 1 y 2. A continuación se realiza una breve explicación de cada una de estas partes.

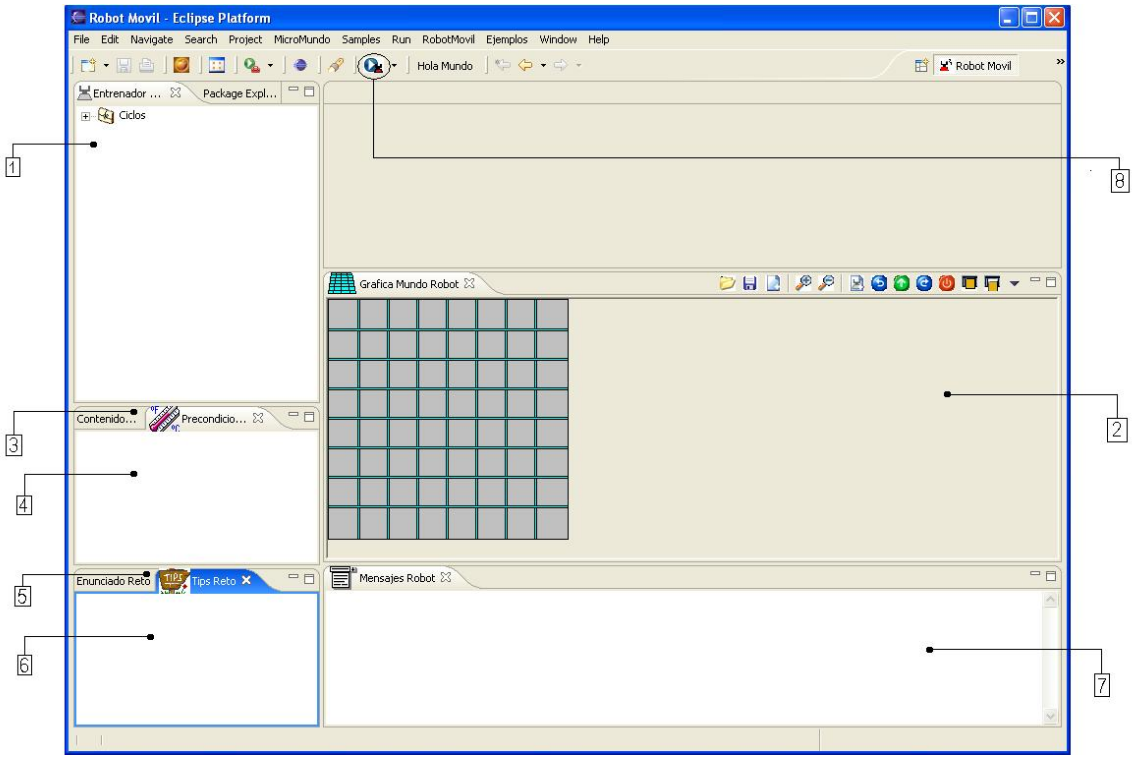


Figura 2. Partes de la perspectiva

1. Explorador del entrenador
2. Vista de la gráfica del mundo del robot
3. Vista de los contenidos del nivel
4. Vista de las precondiciones del nivel
5. Vista de los enunciados de los retos
6. Vista de los tips de los retos
7. Vista de los mensajes del robot
8. Botón para la ejecución de las soluciones de los retos predefinidos

3.1 Explorador del entrenador

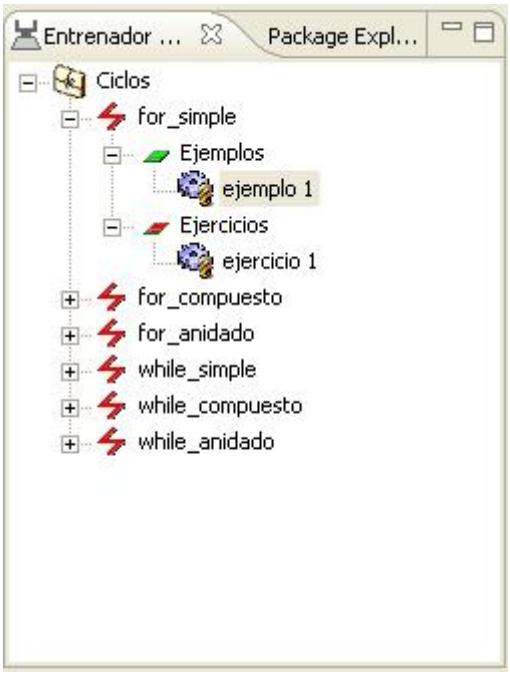


Figura 3. Explorador del robot

Por medio de esta vista se tiene la posibilidad de explorar los diferentes niveles y retos que se encuentran definidos en el entrenador. Cuando se hace click sobre un nivel, se muestran los contenidos y precondiciones asociadas con estos en las vistas respectivas. Cuando se elige un reto de entrenamiento, se despliegan el enunciado y los tips asociados con este en las vistas correspondientes; también se muestra el mundo (en la vista de la gráfica del mundo del robot) relacionado con dicho reto. Además, por medio de esta vista se pueden crear y ejecutar soluciones para los retos predefinidos (ver secciones 4.1 y 4.2). En la figura 3 se muestra la vista explorador del robot.

3.2 Vista de la gráfica del mundo del robot

En esta vista se despliega la gráfica que representa el mundo del robot. Además, permite la manipulación del mundo ya que brinda la posibilidad, entre opciones, de crear robot, muros, cubos, y de modificar el estado de los robots que existen (mover, girar, prender, cambiar de velocidad).

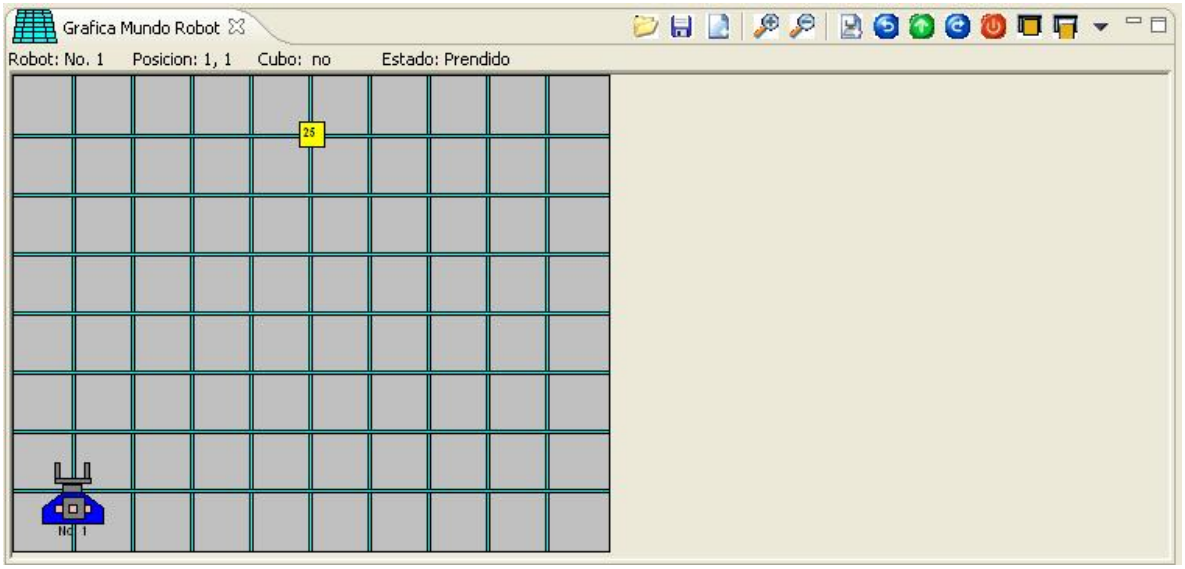


Figura 4. Vista de la gráfica del mundo del robot

3.2.1 Principales características y restricciones del mundo del robot

El mundo del robot es una cuadricula rectangular. El alto y ancho se encuentran representados en el número de cuadros que la cuadricula tenga a lo largo del eje "y" y "x" respectivamente. Por la dimensión máxima que puede tener la cuadricula es (99,99) y la menor (2,2). La posición (1,1) corresponde a la esquina inferior izquierda de la cuadricula, de forma similar a un plano cartesiano.

Un robot puede moverse a través de la cuadrícula, siempre y cuando no se encuentre un muro, otro robot o límite del mundo en su camino. Los robots pueden pasar por encima de los cubos, y si en la posición en la que se encuentran hay un cubo pueden tomarlo. Un robot solo puede llevar un cubo a la vez. Los robots poseen una posición (x, y), una dirección de acuerdo a la orientación en la que se encuentren (la cual puede ser norte, sur, oriente, u occidente) y un color.

En un mundo puede haber más de un robot. Para modificar el estado de un robot, basta con hacer click sobre este. El estado de los robots es mostrado en la barra de estado de la vista (ver figura 5).

Figura 5. Barra de estado del robot. En esta se indica el número del robot (el cual corresponde a la posición en que este fue creado) seleccionado actualmente, la posición en la que se encuentra, si lleva cubo y si se encuentra prendido.

3.2.2 Botones de la vista

La vista cuenta con 12 botones, cuya funcionalidad se explica a continuación.

- Cargar Mundo

  Permite el despliegue en la gráfica de un mundo descrito en un archivo. Cuando se hace click sobre este botón se despliega un dialogo de archivo, como el que se muestra en la figura 6, el cual sólo permite abrir archivo cuya extensión es ".world". Dicho dialogo de archivo se abre por defecto en la ruta `c:\temp\Mundos`; en el caso de que dicha ruta no exista se abre en la carpeta "Mis documenetos".

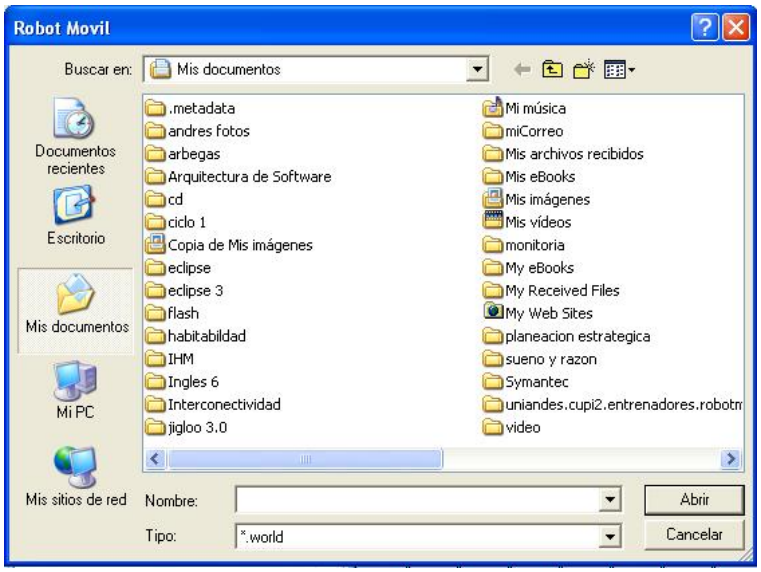


Figura 6. Dialogo para cargar un archivo ".world"

Guardar Mundo

Permite guardar el estado del mundo actual en un archivo con extensión ".world". Cuando se hace click sobre este botón se despliega un dialogo de salvar como el que se muestra en la figura 7.

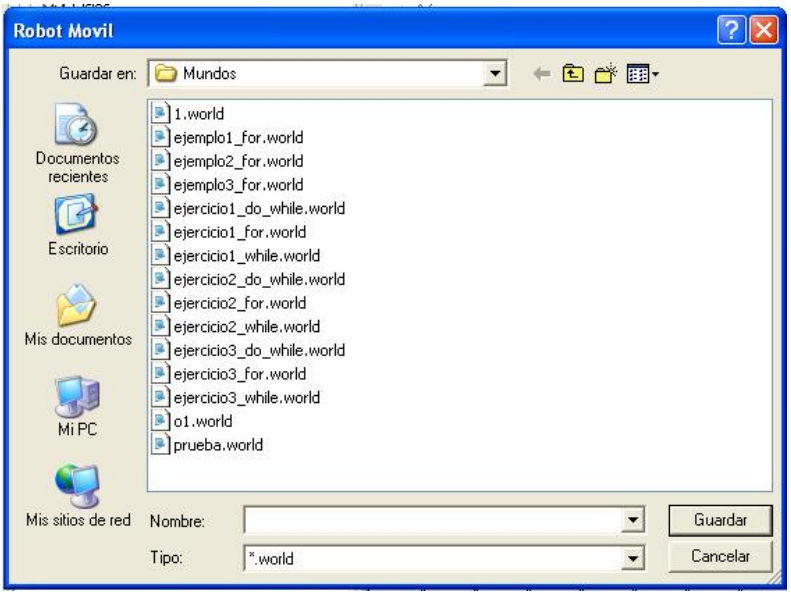


Figura 7. Dialogo para guardar el estado de un mundo en un archivo ".world"

Limpiar

Al hacer click sobre este botón se borran todos los muros, robots y cubos existentes en el mundo que se encuentra cargado en la gráfica.

Aumentar (*Zoom in*)

Realiza *zoom in* sobre la gráfica del robot. Al hacer click sobre este botón se aumenta la escala con la cual es pintada la gráfica del mundo.

Reducir (*Zoom out*)

Realiza *zoom out* sobre la gráfica del robot. Al hacer click sobre este botón se reduce la escala con la cual es pintada la gráfica del mundo.

Nuevo Robot

Permite la creación de un robot. Al hacer click sobre este botón se despliega una ventana como la que se muestra en la figura 8. Existen dos opciones para la creación de un robot: con parámetros y sin parámetros. Cuando se elige la primera opción, es necesario especificar el color, el nombre, la posición (x, y), la orientación y la velocidad (cuyo valor mínimo es 1 y máximo es 9999).

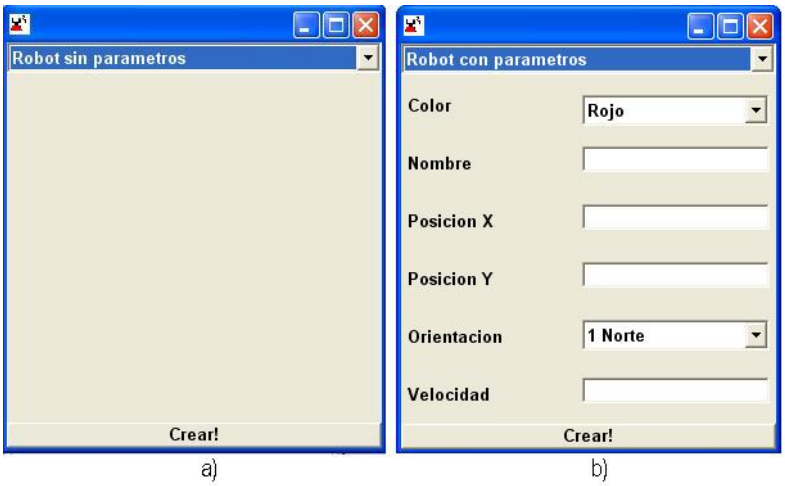


Figura 8. Ventana para la creación de un robot: a) Robot sin parámetros. b) Robot con parámetros-

Al escoger la opción sin parámetros se genera un robot en la posición (1,1), de color azul, orientado hacia el norte con velocidad 10 y nombre C3PO.

Izquierda

Gira 90º a la izquierda el robot que se encuentra actualmente seleccionado. Si no existe ningún robot en el mundo, oprimir este botón no tiene efecto alguno.

Mover Robot

Desplaza el robot una posición en la dirección en la que se encuentra. Si al frente del robot hay un límite del mundo o un muro, el robot se apaga. Si no existe ningún robot en el mundo, oprimir este botón no tiene efecto alguno.

Derecha

Gira 90º a la derecha el robot que se encuentra actualmente seleccionado. Si no existe ningún robot en el mundo, oprimir este botón no tiene efecto alguno.

Prender

Enciende el robot que se encuentra actualmente seleccionado. En caso de que el robot ya se encuentre prendido, oprimir este botón no modifica su estado. Si no existe ningún robot en el mundo, oprimir este botón no tiene efecto alguno.

Tomar Cubo

Hace que el robot recoja el cubo de la posición en la que se encuentra. En el caso de que el robot ya lleve un cubo o no haya cubo en la posición en la que se encuentra se produce un error no fatal. Si no existe ningún robot en el mundo, oprimir este botón no tiene efecto alguno.

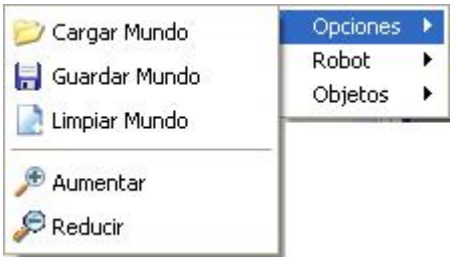
Soltar Cubo

Hace que el robot suelto el cubo que lleva en la posición en la que se encuentra. En el caso de que el robot no lleve un cubo o haya cubo en la posición en la que se encuentra se produce un error no fatal. Si no existe ningún robot en el mundo, oprimir este botón no tiene efecto alguno.

3.2.3 Menús de la vista

La vista de gráfica del mundo del robot se compone de tres menús: “Opciones”, “Robot” y “Objetos”.

Menú Opciones



En este se encuentra los Items de “Cargar Mundo”, “Guardar Mundo”, “Limpiar Mundo“, “Aumentar” y “Reducir”. La función que cumplen estos ítems es idéntica a los botones correspondientes descritos en la sección 3.2.2.

Figura 9. Menú de opciones

Menú Robot

En este menú se encuentran los ítems de “Crear Robot”, “Girar Izquierda”, “Girar Derecha”, “Mover Robot”, “Prender Robot”, “Tomar Cubo” y “Soltar Cubo”. La función que cumplen estos ítems es idéntica a los botones correspondientes descritos en la sección 3.2.2. Adicionalmente se presenta la opción “Cambiar Velocidad”, la cual se describe a continuación.

Figura 10. Menú Robot

- *Cambiar Velocidad*: Modifica la velocidad con la que se mueve el robot. El valor mínimo es 1 y el máximo 9999. Al hacer click sobre este botón se despliega una ventana como la que se muestra en la figura 11.

Figura 11. Ventana para el cambio de velocidad de un robot

Menú Objetos

Este menú presenta los ítems “Crear Muro”, “Crear Cubo” y “Tamaño Cuadricula”.

Figura 12. Menú Objetos

Crear Muro: permite la creación de un muro especificando su longitud, posición y orientación. Al hacer click sobre este ítem se despliega la ventana mostrada en la figura 13.

Figura 13. Ventana para la creación de un muro

Crear Cubo: permite la creación de un cubo especificando su posición, color y valor de la estampa. Este último valor debe ser un entero entre 1 y 99. Al hacer click sobre este ítem se despliega la ventana mostrada en la figura 14.

Figura 14. Ventana para la creación de un cubo

Nuevo tamaño cuadricula: modifica el tamaño de la cuadricula de acuerdo a las dimensiones especificadas. Tanto el valor en "x" como en "y" debe ser un entero entre 2 y 99. Al hacer click sobre este ítem se despliega la ventana mostrada en la figura 15.

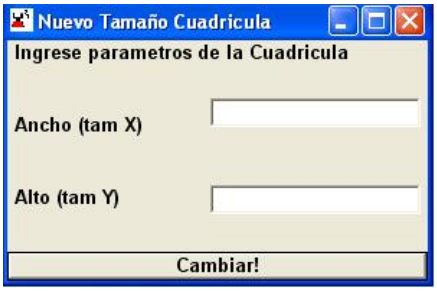
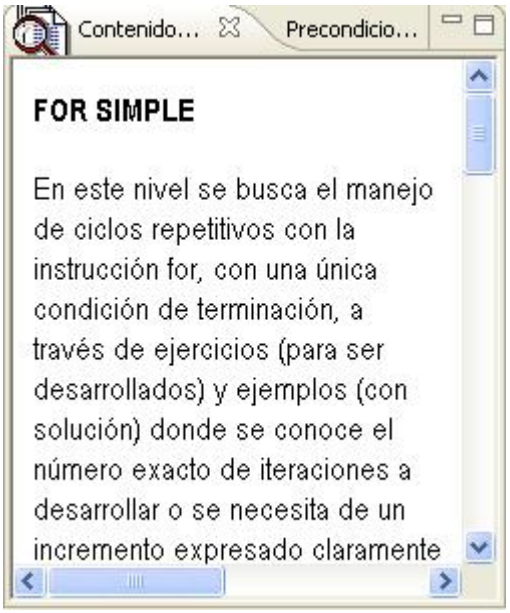


Figura 15. Ventana para cambiar el tamaño de la cuadricula.

3.3 Vista de los contenidos del nivel


Contenido...
Precondicio...
FOR SIMPLE
En este nivel se busca el manejo de ciclos repetitivos con la instrucción for, con una única condición de terminación, a través de ejercicios (para ser desarrollados) y ejemplos (con solución) donde se conoce el número exacto de iteraciones a desarrollar o se necesita de un incremento expresado claramente

Esta vista permite la visualización de los contenidos asociados con un nivel. Para visualizar los contenidos en esta vista es necesario hacer click en el explorador sobre el nivel de interés. En la figura 16 se puede apreciar la vista de contenidos.

Figura 16. Vista de contenidos.

3.4 Vista de las precondiciones del nivel

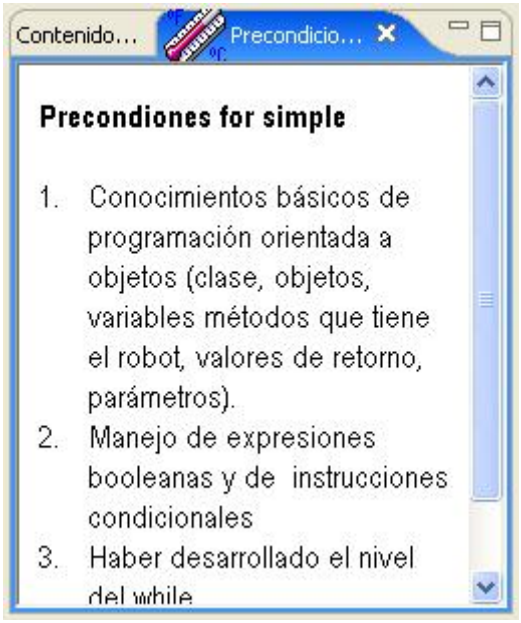

Contenido...
Precondicio...
Precondiones for simple
1. Conocimientos básicos de programación orientada a objetos (clase, objetos, variables métodos que tiene el robot, valores de retorno, parámetros).
2. Manejo de expresiones booleanas y de instrucciones condicionales
3. Haber desarrollado el nivel del while

Figura 17. Vista de precondiciones.

Esta vista permite la visualización de las precondiciones asociadas con un nivel. Para visualizar las precondiciones en esta vista es necesario hacer click en el explorador sobre el nivel de interés. En la figura 17 se puede apreciar la vista de precondiciones.

3.5 Vista de los enunciados de los retos

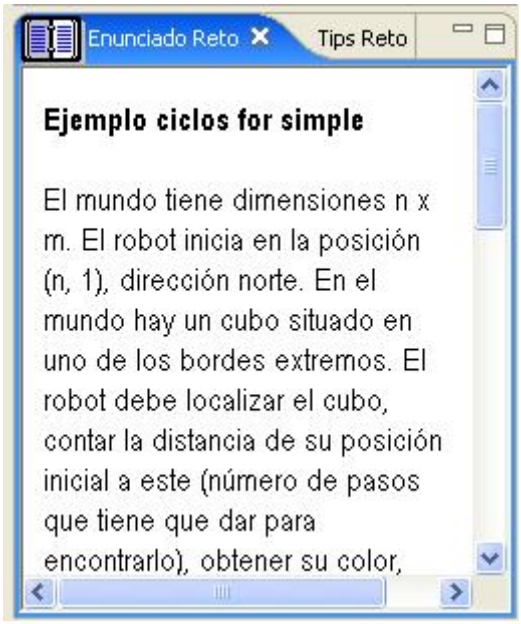
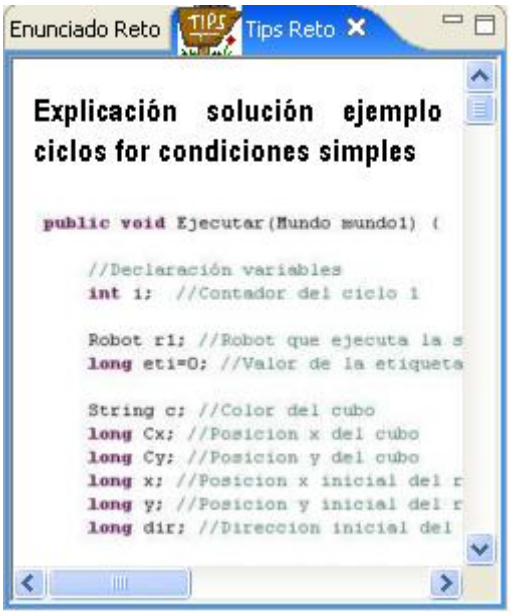

Enunciado Reto
Tips Reto
Ejemplo ciclos for simple
El mundo tiene dimensiones n x m. El robot inicia en la posición (n, 1), dirección norte. En el mundo hay un cubo situado en uno de los bordes extremos. El robot debe localizar el cubo, contar la distancia de su posición inicial a este (número de pasos que tiene que dar para encontrarlo), obtener su color,

Figura 18. Vista de enunciados.

Esta vista permite la visualización del enunciado asociado con un reto de entrenamiento. Para visualizar el enunciado de un reto en esta vista es necesario hacer click en el explorador sobre el ejemplo o ejercicio de interés. En la figura 18 se puede apreciar la vista de precondiciones.

3.6 Vista de los tips de los retos


Enunciado Reto
Tips Reto
Explicación solución ejemplo ciclos for condiciones simples
public void Ejecutar(Mundo mundo1) {
//Declaración variables
int i; //Contador del ciclo 1
Robot r1; //Robot que ejecuta la s
long eti=0; //Valor de la etiqueta
String c; //Color del cubo
long Cx; //Posicion x del cubo
long Cy; //Posicion y del cubo
long x; //Posicion x inicial del r
long y; //Posicion y inicial del r
long dir; //Direccion inicial del

Esta vista permite la visualización de los tips asociados con un reto de entrenamiento. En el caso de los ejemplos, se trata de una explicación de la solución y en el caso de los ejercicios de ayudas para el desarrollo de los mismos. Para visualizar los tips de un reto en esta vista es necesario hacer click en el explorador sobre el ejemplo o ejercircio de interés. En la figura 19 se puede apreciar la vista de tips.

Figura 19. Vista de tips.

3.7 Vista de los mensajes del robot

En esta vista se despliegan los mensajes asociados con las instrucciones y errores que arrojan las operaciones realizadas sobre los robots, y en general sobre el mundo.

Figura 20. Vista de mensajes

3.8 Botón para la ejecución de las soluciones de los retos predefinidos

Figura 21. Botón para correr las soluciones de retos predefinidos.

Este botón permite la ejecución de las últimas soluciones que se hayan corrido por medio del explorador (ver sección 4.2). El hacer click sobre el botón hace que se ejecute la última solución que se haya corrido. Si se hace click sobre la flecha se despliegan

las 6 soluciones que se hayan ejecutado más recientemente. Si no se ha ejecutado previamente ninguna solución, hacer click sobre este botón no tendrá ningún efecto. El item "Ejecutar Solución" del menú "RobotMovil" de la barra de menús de Eclipse cumple con la misma función que este botón.

3.9 Vistas adicionales

La perspectiva Robot Movil utiliza dos vistas de Java. La primera de ellas, es el package explorer (ver figura 22a) que permite la exploración de los proyectos que se crean como proyectos Java. La otra vista es el editor, también de Java, que permite que el estudiante desarrolle la solución de los retos planteados (ver figura 22b).

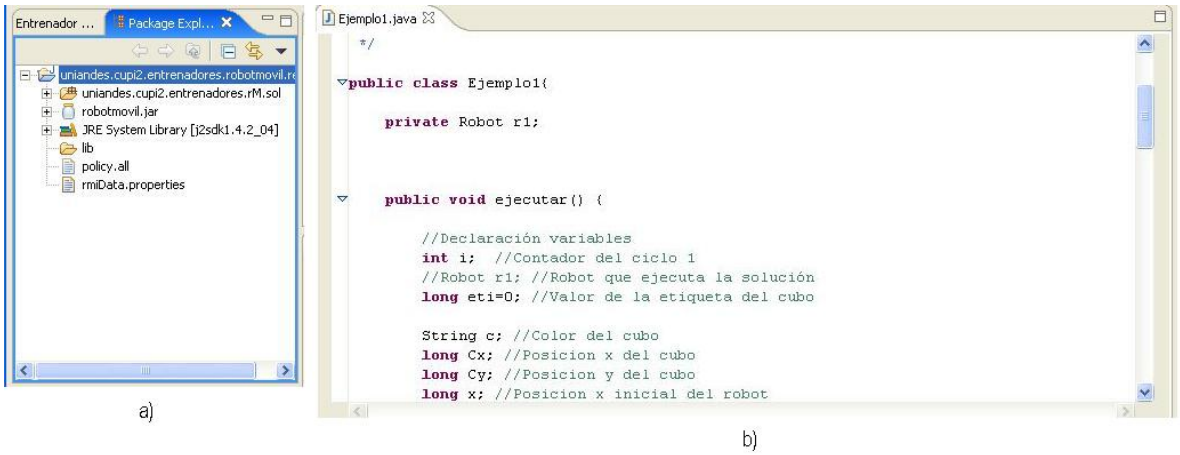


Figura 22. Vistas adicionales de la perspectiva: a) Package explorer. b) Editor

4. TRABAJANDO CON LOS RETOS DE ENTRENAMIENTO PREDEFINIDOS

Aparte de visualizar los enunciados y tips asociados con un reto de entrenamiento es posible crear, desarrollar y ejecutar las soluciones asociadas con estos.

4.1 Creando una nueva solución

Para la creación de una solución asociada con un ejemplo o ejercicio siga los pasos que se enuncian a continuación:

1. Sitúese en la vista explorador del entrador sobre el reto al que desea crearle una solución.
2. Haga click derecho sobre este. Le debe aparecer un menú como el que se muestra en la figura 23a.
3. Elija la opción "Crear Solución", haciendo click sobre esta. Le debe aparecer una ventana como la que se muestra en la figura 23b.
4. Digite el nombre del proyecto y del archivo.
5. Haga click en "Crear!".

Si el proyecto que digita no existe, este es creado. En el caso de que exista, el archivo es creado dentro de este. Cuando ingresa un proyecto y archivo que ya existen pero que no se encuentran

asociados con el reto sobre el que está creando la solución, dicho archivo es asociado con el reto (sin borrarlo ni volverlo a crear).

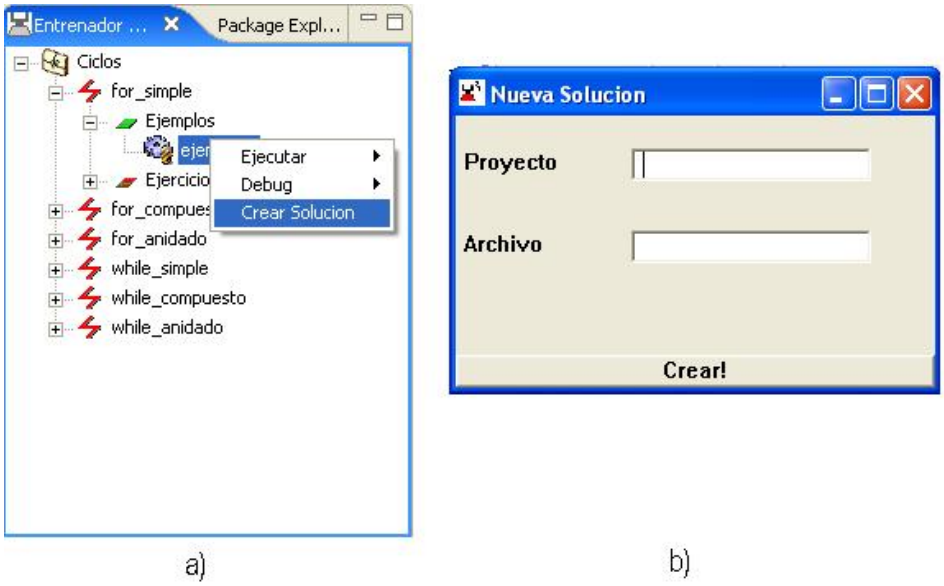


Figura 23. Componentes involucrados en la creación de una solución: a) Explorador del robot. b) Ventana para la creación de una solución

4.2 Ejecutando la solución de un reto

Para la ejecución de una solución asociada con un reto siga las siguientes instrucciones:

1. Sitúese en la vista explorador del entrador sobre el reto al que desea ejecutarle una solución.
2. Haga click derecho sobre este. Le debe aparecer un menú como el que se muestra en la figura 24.
3. Al situarse sobre la opción “Ejecutar” le deben aparecer todas las soluciones asociadas con el reto (ver figura 24).
4. Haga click sobre la solución que desee ejecutar. La ejecución debería iniciar inmediatamente.

A los ejemplos se les crea por defecto un archivo que contiene su solución en un proyecto bajo el nombre “uniandes.cupi2.entrenadores.robotmovil.retos”, de manera que estos siempre deberían tener una solución asociada. Sin embargo, usted también puede crear nuevas soluciones a los ejemplos (ver sección 4.1) o modificar las ya existentes. Con respecto a los ejercicios, es necesario crearles la solución (ver sección 4.1).

Una vez se ejecuta una solución de la forma expuesta en este apartado, es posible también correrlas por medio del botón para ejecutar soluciones explicado en la sección 3.8.

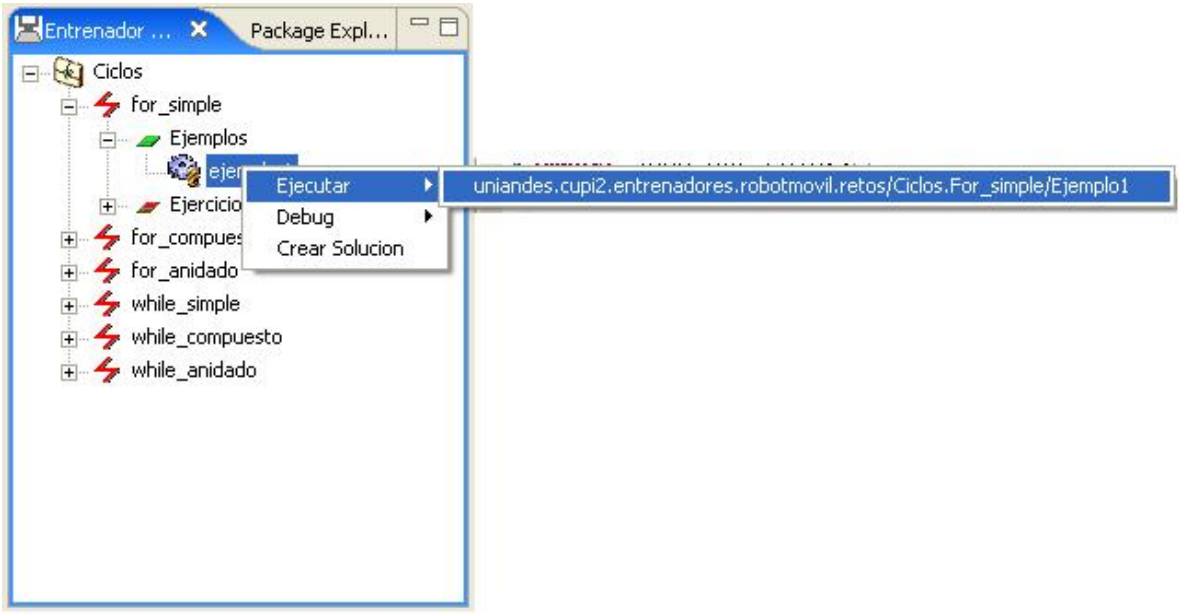


Figura 24. Forma de ejecutar una solución de un reto predefinido

4.3 Haciendo debug a la solución de un reto

Para la realización de debug sobre una solución siga los siguientes pasos:

1. Sitúese sobre el archivo solución sobre el que desea hacer debug.
2. Ponga un break point en la línea desde la que desea hacer el debug.
3. Sitúese en la vista explorador del entrador sobre el reto al que desea hacerle debug a una de sus soluciones.
4. Haga click derecho sobre este. Le debe aparecer un menú como el que se muestra en la figura 25a.
3. Al situarse sobre la opción “Debug” le deben aparecer todas las soluciones asociadas con el reto (ver figura 25a). Si el reto no tiene soluciones asociados, debe crear y desarrollarle una.
4. Haga click sobre la solución sobre la que desee hacer debug. Le debe aparecer un diálogo (ver figura 25b) preguntándole si desea cambiar de perspectiva, elija “yes”.
5. A continuación se debe abrir una perspectiva como la que se muestra en la figura 26.
6. Para avanzar en la solución paso a paso utilice el botón , es decir, el botón “Step Over”.

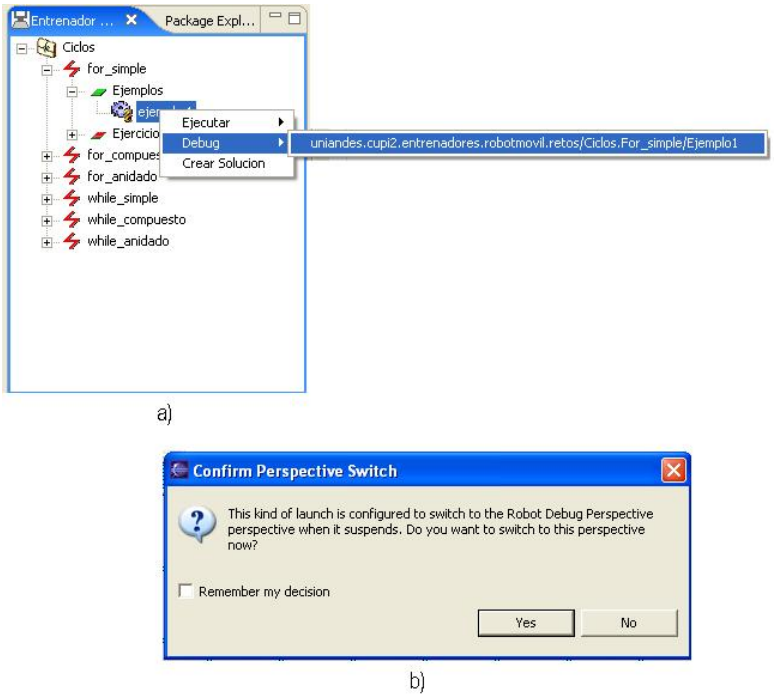


Figura 25. Componentes involucrados en el debug de una solución:
a) Entrenador Explorer. b) Dialogo para cambio de perspectiva.

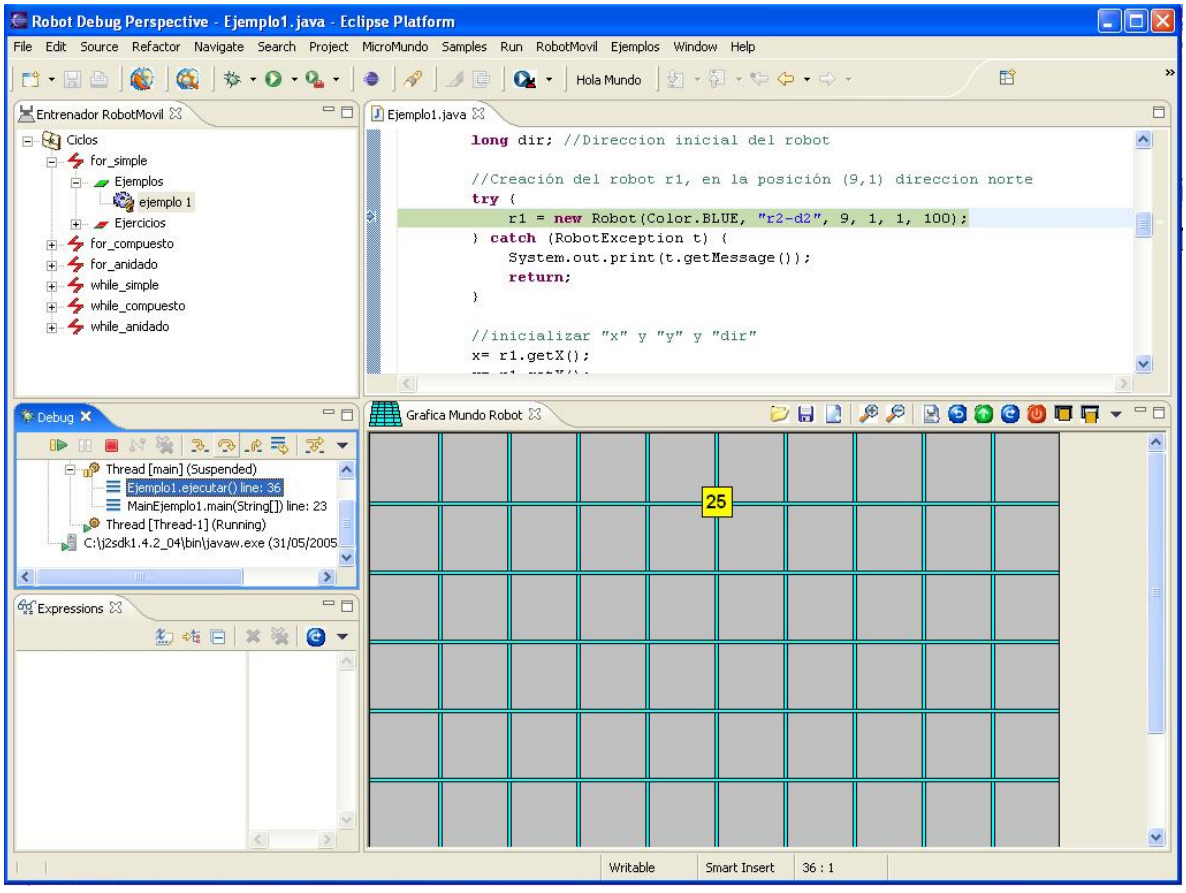


Figura 26. Perspectiva de Debug del Robot.

4.4 resolviendo problemas con la ejecución de soluciones

Cuando no pueda ejecutar las soluciones del entrenador, esto puede deberse a problemas con el puerto en el que éstas están siendo lanzadas. Por está razón es necesario cambiar el puerto en el que las soluciones son ejecutadas, por medio de las preferencias del proyecto (ver sección 6 de este manual).

5. CREANDO UN NUEVO PROYECTO

Para la creación de un proyecto nuevo para el desarrollo de soluciones del robot (no necesariamente asociadas con los retos predefinidos) siga los pasos siguientes:

1. Elija `File>New>Other`. Le debe aparecer una ventana similar a la que se muestra a continuación:

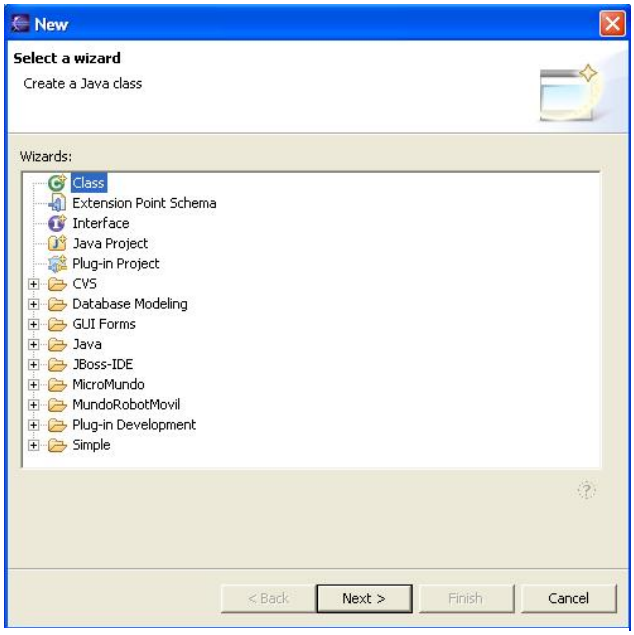
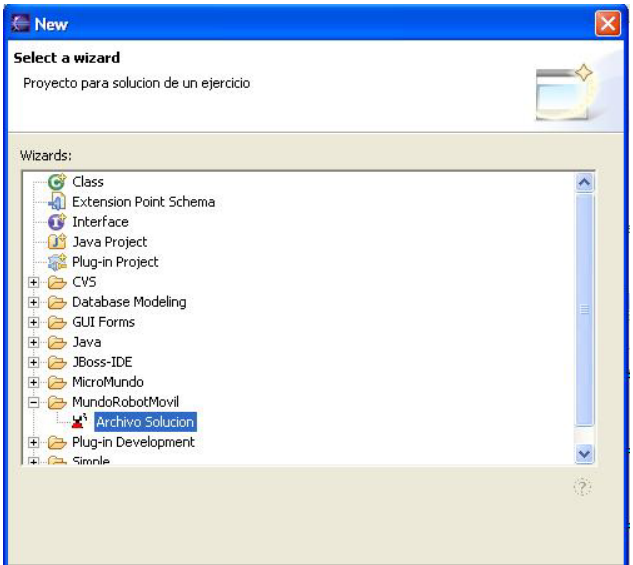


Figura 27. Ventana inicial para la creación de un nuevo

2. Sitúese en la carpeta “MundoRobotMovil” y elija “Archivo Solucion”. Haga click en “Next >”.



3. A continuación le debe aparecer una ventana como la siguiente:

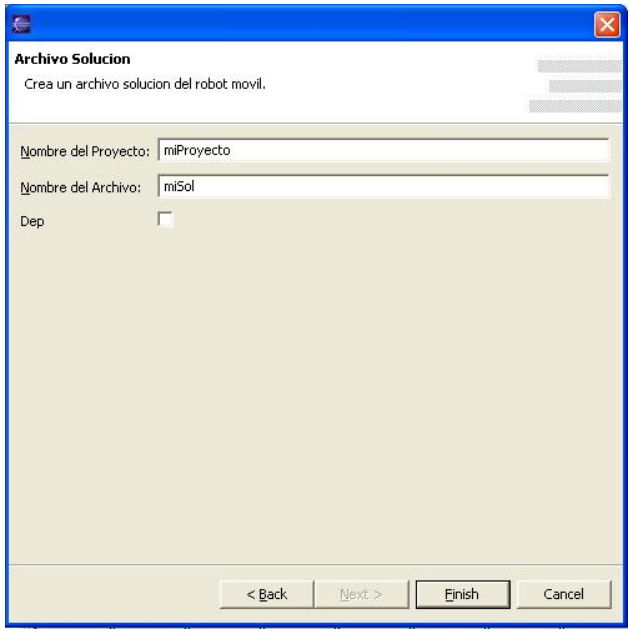


Figura 29. Ventana para la creación del proyecto

4. Ingrese el nombre del proyecto y del archivo y haga click en “Finish”.
5. En el editor se debe abrir un archivo con el nombre que introdujo en el paso 4.

6. CAMBIANDO LAS PREFERENCIAS DEL PROYECTO

El proyecto maneja dos preferencias: una para el número del puerto en el que se ejecutan las soluciones y otra para el nombre del archivo en el que se crean las soluciones por defecto de los ejemplos. El valor del puerto es un entero entre 0 y 49000; este valor solo deberá cambiarse en caso de que la aplicación indique que el puerto en que se trata de correr las soluciones se encuentra ocupado.

El cambio de nombre del proyecto en el que se crean las soluciones por defecto hace que se cree un proyecto con el nuevo nombre con las soluciones de los ejemplos a medida que estos van siendo seleccionados en el explorador del robot. El cambio del valor de esta preferencia no ocasiona que se borre ni renombre el proyecto que se había creado con el nombre anterior.

Para modificar alguna de las preferencias del proyecto siga los siguientes pasos:

1. Elija `Windows>Preferences`. Le debe aparecer una ventana similar a la siguiente:

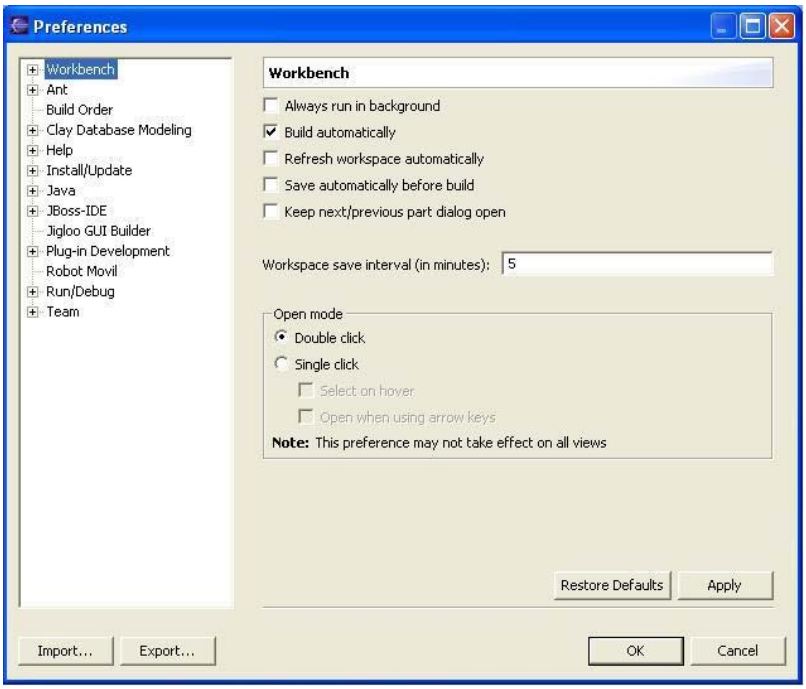


Figura 30. Ventana inicial preferencias

2. Sitúese en “Robot Movil”. Le debe aparecer una ventana similar a la que se muestra en la figura 31.

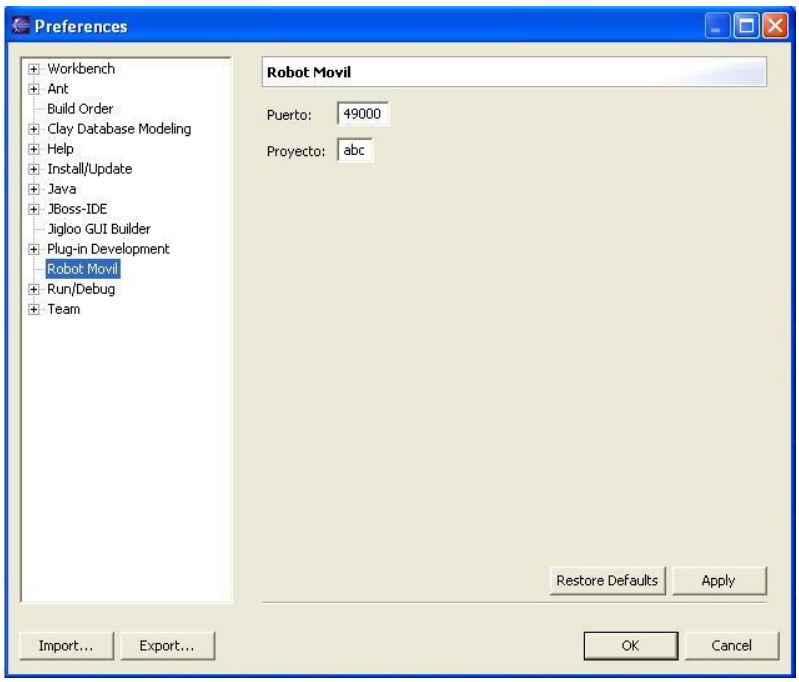


Figura 31. Ventana de las preferencias del plugin

3. Modifique el valor de una o ambas preferencias. Haga click en “Apply” y “Ok” o solo en “Ok”.
4. El valor de las preferencias fue modificado.

Si desea restaurar los valores por defecto de ambas preferencias, haga click en “Restore Defaults”, y luego haga click en “Apply”.

7. API DEL ROBOT

---

uniandes.cupi2.robotMovil.kernelCliente

Class Robot

```
java.lang.Object
  └ uniandes.cupi2.robotMovil.kernelCliente.Robot
```

```
public class Robot
extends java.lang.Object
```

Author:

Cesar Restrepo- Daniel Romero

Constructor Summary

Robot() Constructor sin parámetros
Robot(java.awt.Color col, java.lang.String nom, long X, long Y, long sent, long veloc) Constructor con parámetros

Method Summary

void	**agregarMensaje**(java.lang.String mensaje) pre: TRUE post: se agrega un mensaje al robot
boolean	**cargandoCube**() pre: TRUE post: se retorna TRUE si el robot tiene un cubo, y FALSE si no.
java.awt.Color	**colorDelCuboCargado**() pre: el robot esta cargando un cubo post: retorna el color del cubo que esta cargando el robot post: se retorna el color del cubo que está cargando el robot
java.lang.String	**colorDelCuboCargadoCadena**()

	pre: El robot lleva un cubo post: obtiene el color del cubo, como String
boolean	**contactoConCubo** () pre:TRUE post: retorna TRUE si en la posición del robot hay un cubo o FALSE de lo contrario
void	**definirVelocidad**(long velocidad) pre: TRUE post: se le asigna al robot la velocidad especificada, todos los robots del mundo deben tener la misma velocidad
void	**soltarCubo**() pre: el robot está cargando un cubo post: se cambian del robot sus atributos y asociaciones de manera que ya no esté cargando un cubo
long	**obtenerDireccion**() pre: TRUE post: retorna el entero correspondiente a la orientación del robot
long	**obtenerX**() pre: TRUE post: retorna la coordenada x en la que se encuentra el robot
long	**obtenerY**() pre: TRUE post: retorna la coordenada y en la que se encuentra el robot
boolean	**tieneCuboAlFrente**() pre: TRUE post: retorna TRUE si en la posición exactamente adelante del robot hay un cubo, o FALSE de lo contrario
boolean	**tieneCuboALaIzquierda**() pre: TRUE post: retorna TRUE si en la posición exactamente a la izquierda del robot hay un cubo, o FALSE de lo contrario
boolean	**tieneCuboALaDerecha**() pre: TRUE post: retorna TRUE si en la posición exactamente a la derecha del robot hay un cubo, o FALSE de lo contrario
boolean	**tieneRobotAlFrente**() pre: TRUE post: retorna TRUE si en la posición exactamente adelante del robot hay otro robot, o FALSE de lo contrario
boolean	**tieneRobotALaIzquierda**() pre: TRUE post: retorna TRUE si en la posición exactamente a la

	izquierda del robot hay otro robot, o FALSE de lo contrario
`boolean`	**`tieneRobotALaDerecha`**`()` pre: TRUE post: retorna TRUE si en la posición exactamente a la derecha del robot hay otro robot, o FALSE de lo contrario
`boolean`	**`tieneMuroAlFrente`**`()` pre: TRUE post: retorna TRUE si en la posición exactamente adelante del robot hay un muro, o FALSE de lo contrario
`boolean`	**`tieneMuroALaIzquierda`**`()` pre: TRUE post: retorna TRUE si en la posición exactamente a la izquierda del robot hay un muro, o FALSE de lo contrario
`boolean`	**`tieneMuroALaDerecha`**`()` pre: TRUE post: retorna TRUE si en la posición exactamente a la derecha del robot hay un muro, o FALSE de lo contrario
`boolean`	**`estaPrendido`**`()` pre: TRUE post: se retorno FALSE si el robot esta prendido, o FALSE de lo contrario
`void`	**`mover`**`()` pre: el robot no tiene muro al frente post: el robot se mueve una posición hacia la dirección de su orientación
`boolean`	**`limiteAlFrente`**`()` pre: TRUE post: retorna TRUE si en la posición exactamente adelante del robot esta el limite del mundo, o FALSE de lo contrario
`boolean`	**`limiteALaIzquierda`**`()` pre: TRUE post: retorna TRUE si en la posición exactamente a la izquierda del robot esta el limite del mundo, o FALSE de lo contrario
`boolean`	**`limiteALaDerecha`**`()` pre: TRUE post: retorna TRUE si en la posición exactamente a la derecha del robot esta el limite del mundo, o FALSE de lo contrario
`java.awt.Color`	**`colorDelRobot`**`()` pre: TRUE post: el robot retorna el color
`java.lang.String`	**`nombreDelRobot`**`()` pre: TRUE

	post: el robot retorna el nombre
void	**rotarALaIzquierda**() pre: TRUE post: el robot gira 90 grados a su izquierda
void	**rotarALaDerecha**() pre: TRUE post: el robot gira 90 grados a su derecha
long	**marcaDelCuboCargado**() pre: el robot esta cargando un cubo post: se retorna la carga del cubo que esta cargando el robot
void	**tomarCubo**() pre: el robot no esta cargando un cubo post: se cambian del robot sus atributos y asociaciones de manera que tenga un cubo asociado
void	**prender**() pre: TRUE post: el robot se prende y queda con la capacidad de realizar acciones en el mundo

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Robot

```
public Robot()
```
throws RobotException

Constructor sin parametros

Throws:

RobotException

Robot

```
public Robot(java.awt.Color col,
             java.lang.String nom,
             long X,
             long Y,
```

```
        long sent,
        long veloc)
throws RobotException
```

Constructor con parametros

Parameters:

`col` - = Color del robot

`nom` - = Nombre del robot

`X` - = Cordenada X del robot.

`Y` - = Cordenada Y del robot

`sent` - = Sendtido del nuevo robot

`veloc` - = Velocidad del robot

Throws:

`RobotException`

Method Detail

cargandoCubo

```
public boolean cargandoCubo()
```

pre: TRUE
post: se retorna TRUE si el robot tiene un cubo, y FALSE si no./p>

Returns:

Booleano que indica si el robot lleva un cubo o no

---

colorDelCuboCargado

```
public java.awt.Color colorDelCuboCargado()
```

pre: el robot esta cargando un cubo post: retorna el color del cubo que esta cargando el robot
post: se retorna el color del cubo que está cargando el robot

Returns:

El color del cubo que lleva el robot

---

contactoConCubo

```
public boolean contactoConCubo()
```

pre:TRUE
post: retorna TRUE si en la posición del robot hay un cubo o FALSE de lo contrario

Returns:

Booleano que indica si hay un cubo en la posición del robot o no

---

obtenerDireccion

`public long obtenerDireccion()`

pre: TRUE
post: retorna el entero correspondiente a la orientación del robot
Returns:
Entero correspondiente a la orientación del robot

---

obtenerX

`public long obtenerX()`

pre: TRUE
post: retorna la coordenada x en la que se encuentra el robot
Returns:
Entero que corresponde a la coordena x en la que se encuentra el robot

---

obtenerY

`public long obtenerY()`

pre: TRUE
post: retorna la coordenada y en la que se encuentra el robot
Returns:
Entero que corresponde a la coordena y en la que se encuentra el robot

---

tieneRobotAlFrente

`public boolean tieneRobotAlFrente()`

pre: TRUE
post: retorna TRUE si en la posición exactamente adelante del robot hay otro robot o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene otro robot al frente

---

tieneRobotALaIzquierda

`public boolean tieneRobotALaIzquierda()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la izquierda del robot hay otro robot o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene otro robot a la izquierda

---

tieneRobotALaDerecha

`public boolean` **`tieneRobotALaDerecha`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la derecha del robot hay otro robot o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene otro robot a la derecha

---

tieneCuboAlFrente

`public boolean` **`tieneCuboAlFrente`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente adelante del robot hay un cubo o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un cubo al frente

---

tieneCuboALaIzquierda

`public boolean` **`tieneCuboALaIzquierda`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la izquierda del robot hay un cubo o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un cubo a la izquierda

---

tieneCuboALaDerecha

`public boolean` **`tieneCuboALaDerecha`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la derecha del robot hay un cubo o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un cubo a la derecha

---

tieneMuroAlFrente

`public boolean` **`tieneMuroAlFrente`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente adelante del robot hay un muro o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un muro al frente

---

tieneMuroALaIzquierda

`public boolean` **`tieneMuroALaIzquierda`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la izquierda del robot hay un muro o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un muro a la izquierda

---

tieneMuroALaDerecha

`public boolean` **`tieneMuroALaDerecha`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la derecha del robot hay un muro o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un muro a la derecha

---

limiteAlFrente

`public boolean` **`limiteAlFrente`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente adelante del robot esta el limite del mundo o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un límite del mundo al frente

---

limiteALaIzquierda

`public boolean` **`limiteALaIzquierda`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la izquierda del robot esta el limite del mundo o FALSE de lo contrario
Returns:
Booleano que indica si el robot tiene un límite del mundo a la izquierda

---

limiteALaDerecha

`public boolean` **`limiteALaDerecha`**`()`

pre: TRUE
post: retorna TRUE si en la posición exactamente a la derecha del robot esta el limite del mundo o FALSE de lo contrario

Returns:
Booleano que indica si el robot tiene un límite del mundo a la derecha

---

marcaDelCuboCargado

`public long` **`marcaDelCuboCargado`**`()`

pre: el robot esta cargando un cubo
post: se retorna la carga del cubo que esta cargando el robot
Returns:
Long que indica el valor de la estampa del cubo que lleva el robot

---

agregarMensaje

`public void` **`addMessage`**`(java.lang.String Mensaje)`

pre: TRUE
post: se agrega un mensaje al robot
Parameters:
`Message=` - Mensaje para imprimir en la vista de mensajes del robot

---

definirVelocidad

`public void` **`definirVelocidad`**`(long velocidad)`

pre: TRUE
post: se le asigna al robot la velocidad especificada, todos
los robots del mundo deben tener la misma velocidad
Parameters:
`Speed=` - Long que define la nueva velocidad del robot

---

soltarCubo

`public void` **`soltarCubo`**`()`

pre: el robot esta cargando un cubo
post: se cambian del robot sus atributos y asociaciones de manera
que ya no este cargando un cubo

---

mover

`public void` **`mover`**`()`

pre: el robot no tiene muro al frente
post: el robot se mueve una posición hacia la dirección de
su orientación

---

rotarALaIzquierda

`public void` **`rotarALaIzquierda`**`()`

pre: TRUE
post: el robot gira 90 grados a su izquierda

---

rotarALaDerecha

`public void` **`rotarALaDerecha`**`()`

pre: TRUE
post: el robot gira 90 grados a su derecha

---

tomarCubo

`public void` **`tomarCubo`**`()`

pre: el robot no esta cargando un cubo
post: se cambian del robot sus atributos y asociaciones
de manera que tenga un cubo asociado

---

prender

`public void` **`prender`**`()`

pre: TRUE
post: el robot se prende y queda con la capacidad de realizar
acciones en el mundo

---

estaPrendido

`public boolean` **`isTurnOn`**`()`

pre: TRUE
post: se retorno FALSE si el robot esta prendido o FALSE
de lo contrario
Returns:
Booleano que indica el estado del robot

---

nombreDelRobot

`public java.lang.String` **`robotName`**`()`

pre: TRUE
post: el robot retorna el nombre
Returns:

String con el nombre del robot

---

colorDelRobot

```
public java.awt.Color robotColor()
```

pre: TRUE
post: el robot retorna el color
Returns:
Color del robot

---

colorDelCuboCargadoCadena

```
public java.lang.String colorCubeCarriedString()
```

pre: El robot lleva un cubo
post: obtiene el color del cubo, como String
Returns:
String que representa el nombre del color del cubo

---

8. CRÉDITOS

Autores Plugin: Daniel Francisco Romero Acero
Cesar Augusto Restrepo Sarmiento

Asesores de Tesis: Katalina Marcos
Carlos Noguera

Autores de la aplicación original: Pablo Barvo
Diana Puentes
Sandra Ruiz
Manuel Muñoz
Christian Kraus

Modificaciones sobre la aplicación original: Daniel Francisco Romero Acero
Cesar Augusto Restrepo Sarmiento

Autor manual: Daniel Francisco Romero Acero, Cesar Augusto Restrepo Sarmiento

9. REFERENCIAS

1. Quiroga J. (1998). Tesis de Grado- Manual del Usuario.